



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение  
высшего образования

**"МИРЭА - Российский технологический университет"**

**РТУ МИРЭА**

---

**Институт Информационных Технологий**

**Кафедра Вычислительной Техники**

**ПРАКТИЧЕСКАЯ РАБОТА**

**по дисциплине**

**«Системный анализ данных СППР»**

Студент группы: ИКБО-42-23

Голев С.С.  
(Ф. И.О. студента)

Преподаватель

Железняк Л.М.  
(Ф.И.О. преподавателя)

Москва 2025

## СОДЕРЖАНИЕ

ВВЕДЕНИЕ .....	3
6 АЛГОРИТМ ОБЕЗЬЯН .....	4
6.1 Цель и задачи практической работы .....	4
6.2 Постановка задачи .....	4
6.3 Ручной расчёт .....	5
6.4 Результат работы.....	7
6.5 Результат работы нахождения кратчайшего пути .....	8
ЗАКЛЮЧЕНИЕ .....	9
СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ .....	10
ПРИЛОЖЕНИЯ.....	11

## **ВВЕДЕНИЕ**

Алгоритм обезьян относится к классу роевых методов оптимизации, вдохновлённых поведением обезьян при поиске пищи и перемещении по деревьям. Основная идея заключается в моделировании процессов лазания, прыжков и наблюдения, которые позволяют агентам исследовать пространство решений. Обезьяны коллективно обмениваются информацией о найденных «лучших» позициях, что способствует направленному поиску глобального оптимума. В работе рассматривается применение алгоритма обезьян для нахождения минимума функции. Такой подход сочетает глобальный и локальный поиск, обеспечивая баланс между исследованием новых областей и уточнением найденных решений. В результате алгоритм обезьян демонстрирует способность эффективно обходить локальные минимумы и находить высококачественные приближённые решения сложных задач оптимизации.

## 6 АЛГОРИТМ ОБЕЗЬЯН

Алгоритм основан на поведении стаи обезьян, они перемещаются с помощью прыжков в окрестностях своего местоположения и если не находят лучшего значения, то перемещаются к центру нахождения всех обезьян, в следствие чего находят точки экстремума функций.

### 6.1 Цель и задачи практической работы

Целью практической работы является изучение принципов алгоритма обезьян и его применение для решения задач нахождения точек экстремума.

Для достижения поставленной цели необходимо выполнить следующие задачи:

1. Выполнить ручной расчёт одной итерации алгоритма обезьян для функции Била;
2. Реализовать алгоритм обезьян на языке Python для автоматического поиска точек экстремума;
3. Применить алгоритм для функции Била;

Таблица 6.1 – Функция Била

Формула	Глобальный минимум	Метод поиска
$f(x, y) = (1.5 - x + xy)^2 + (2.25 - x + xy^2)^2 + (2.625 - x + xy^3)^2$	$f(3, 0.5) = 0$	$-4.5 \leq x, y \leq 4.5$

### 6.2 Постановка задачи

В рамках практической работы необходимо реализовать алгоритм обезьян вручную и кодово, алгоритм будет проверяться на функции Била.

### 6.3 Ручной расчёт

Выполним ручной расчёт одной итерации данного алгоритма на примере функции Била.

Первоначально все обезьяны отправляются на случайные точки.

После каждая обезьяна начинает делать локальные прыжки фиксированное количество раз, если после прыжка, значение функции меньше изначального, количество прыжков обнуляется.

Допустим, что обезьяна попала в точку (2, 1) со значением функции 14.203125, далее необходимо определить область, в которой обезьяна будет совершать прыжки.

$$x_{ij} = ((x_{ij} - b); (x_{ij} + b)) \quad (6.1)$$

Формула 6.1, где  $b$  – длина прыжка, используется вычисления длины прыжка. За  $b$  возьмём 1. По формуле 6.1 вычислим область для первой и для второй координаты.

Для первой координаты  $[2 - 1 = 1; 2 + 1 = 3]$

Для второй координаты  $[1 - 1 = 0; 1 + 1 = 2]$

Если лимит прыжков исчерпан, обезьяна совершает глобальный прыжок в сторону центра всех обезьян, перелетая его.

$$x_{ij} = x_{ij} + \gamma(x_j^c - x_{ij}) \quad (6.2)$$

$$x_j^c = \frac{1}{S} \sum_{i=1}^S x_{ij} \quad (6.3)$$

Формула 6.2 является экстраполяционной функцией, которая рассчитывает куда производится глобальный прыжок,  $x_j^c$  – центральная

точка среди всех обезьян, получаемая по формуле 6.3, где  $S$  – количество обезьян,  $x_{ij}$  – текущее положение обезьяны.  $\gamma$  – случайный параметр в диапазоне от 1 до 2, с помощью которого обезьяна перелетает центр и данные формулы вычисляются как для первой так и для второй координаты.

Данные действия повторяются для всей стаи обезьян пока не будет выполнен критерий остановки.

Перейдём к ручному расчёту одной итерации, возьмём 3 обезьяны с максимальным количеством локальных прыжков равных 5.

Обезьяна 1:  $x = 1.0, y = 2.0, f(x,y) = 126.45$ ;

Обезьяна 1:  $x = 2.0, y = 0.5, f(x,y) = 39.45$ ;

Обезьяна 1:  $x = 3.0, y = 1.0, f(x,y) = 14.20$ ;

По формуле 6.3 рассчитаем центр стаи.

Центр:  $x_c = 0.67$  и  $y_c = 1.17$ ;

Совершим локальные прыжки для первой обезьяны, область прыжка будет рассчитываться по формуле 6.1.

Обезьяна 1:

Прыжок 1: Область прыжка:  $[0.9, 1.1]$  и  $[1.9, 2.1]$ ;

Новое место:  $x = 1.05, y = 1.95, f(x,y) = 120.82$ ;

Это значение лучше, сбрасываем количество оставшихся прыжков до 5 и записываем эти значения в лучший результат.

Прыжок 2: Область прыжка:  $[0.95, 1.15]$  и  $[1.85, 2.05]$ ;

Новое место:  $x = 1.12, y = 1.88, f(x,y) = 112.02$ ;

Это значение лучше, сбрасываем количество оставшихся прыжков до 5 и записываем эти значения в лучший результат.

Прыжок 3: Область прыжка:  $[1.02, 1.22]$  и  $[1.78, 1.98]$ ;

Новое место:  $x = 1.18, y = 1.82, f(x,y) = 103.48$ ;

Это значение лучше, сбрасываем количество оставшихся прыжков до 5 и записываем эти значения в лучший результат.

Прыжок 4: Область прыжка:  $[1.08, 1.28]$  и  $[1.72, 1.92]$ ;

Новое место:  $x = 1.25, y = 1.75, f(x,y) = 94.44$ ;

Это значение лучше, сбрасываем количество оставшихся прыжков до 5 и записываем эти значения в лучший результат.

Прыжок 5: Область прыжка:  $[1.15, 1.35]$  и  $[1.65, 1.85]$ ;

Новое место:  $x = 1.30, y = 1.70, f(x,y) = 87.177$ ;

Это значение лучше, сбрасываем количество оставшихся прыжков до 5 и записываем эти значения в лучший результат.

Данные шаги повторяются, пока не случится 5 прыжков, при которых новое значение функции будет хуже прошлых. Если за всё количество локальных прыжков значение функции не улучшилось, обезьяна совершает глобальный прыжок, иначе остаётся в своей точке. Центр стаи у нас есть это координаты  $x_c$  и  $y_c$ .

Воспроизведём ситуацию, при которой первая обезьяна постоянно совершала бы плохие прыжки и ей пришлось бы делать глобальный прыжок, воспользуемся формулой 6.2 и рассчитаем координаты прыжка.

Тогда обезьяна 1 находится в точке (1,2), центр находится в точке (0.67,1.17) и  $\gamma$  равен 1.5.

Новые значения :  $x = 0.505, y = 0.921$ , данные значения находятся в указанных пределах, поэтому рассчитаем функцию  $f(0.505, 0.921) = 13.174$ .

После расчёта новых координат первой обезьяны, мы переходим к следующей и так пока все обезьяны не перейдут к новым позициям.

## 6.4 Результат работы

Реализуем нахождение точек экстремума с помощью алгоритма обезьян, количество обезьян: 15, максимальное количество локальных

прыжков: 20, шаг 0.1, количество итераций: 1000, выполним реализацию на языке Python. Реализация представлена в приложении Е.

```
(venv) PS C:\Users\semen\Desktop\MIREA\System_data_analysis\Practice6> py .\monkey.py
Эталон:  $f(3, 0.5) = 0$ 
Результат  $x = 2.99504$   $y = 0.49872$   $f = 0.00000401$ 
```

**Рисунок 6.1 – Пример выполнения алгоритма обезьян**

## **6.5 Результат работы нахождения кратчайшего пути**

В ходе практической работы был выполнен ручной расчёт одной итерации алгоритма обезьян.

Данный расчёт демонстрирует работу алгоритма обезьян: обезьяны совершают локальные и глобальные прыжки пока не будет выполнен критерий остановки.

Далее была выполнена кодовая реализация данного алгоритма, которая позволила автоматически находить точки минимума функции;

В результате работы показано, что алгоритм обезьян является эффективным методом для точек экстремума функции и может использоваться в задачах оптимизации.

## **ЗАКЛЮЧЕНИЕ**

В ходе работы был реализован алгоритм обезьян для решения задачи оптимизации. Проведённые эксперименты показали, что данный метод способен эффективно приближаться к глобальному минимуму. Особенностью алгоритма является сочетание фаз лазания, прыжков и наблюдения, что позволяет моделировать баланс между локальным уточнением и глобальным поиском. Благодаря обмену информацией между агентами алгоритм адаптируется к структуре задачи и демонстрирует устойчивость к случайным возмущениям. Полученные результаты подтвердили эффективность и надёжность алгоритма обезьян, а также его применимость для решения широкого спектра сложных задач оптимизации.

## СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

1. Python Software Foundation. Python Documentation — [Электронный ресурс]. URL: <https://docs.python.org/3/> (дата обращения: 15.09.2025).
2. Лутц М. Изучаем Python. 5-е изд. / пер. с англ. — Санкт-Петербург: Символ-Плюс, 2019. — 1648 с.
3. Баляев С. А. Объектно-ориентированное программирование. Учебное пособие. — Москва : ФОРУМ, ИНФРА-М, 2020. — 256 с.
4. Гринберг Д. Программирование на Python 3. Подробное руководство. — Москва : Вильямс, 2014. — 832 с.

## **ПРИЛОЖЕНИЯ**

Приложение Е – Код программы “Алгоритм обезьян”

## Приложение Е

### Код программы Алгоритм обезьян

*Листинг Е.1 — Основной алгоритм программы*

```
import numpy as np

def f(x, y):
    return (1.5 - x + x*y)**2 + (2.25 - x + x*(y**2))**2 + (2.625 -
x + x*(y**3))**2

class Monkey:
    def __init__(self):
        self.x = 0
        self.y = 0
        self.f = 0
        self.xi = 0
        self.yi = 0
        self.fi = 0
        self.step = 0.1

    def calc(self):
        self.f = f(self.x, self.y)
        return self.f

    def make_local_jump(self, max_jump):
        self.xi, self.yi, self.fi = self.x, self.y, self.f
        best_xi, best_yi, best_fi = self.xi, self.yi, self.fi
        jumps_left = max_jump
        while jumps_left > 0:
            trial_x = np.random.uniform(self.xi - self.step,
self.xi + self.step)
            trial_y = np.random.uniform(self.yi - self.step,
self.yi + self.step)
            trial_x = np.clip(trial_x, -4.5, 4.5)
            trial_y = np.clip(trial_y, -4.5, 4.5)
            trial_f = f(trial_x, trial_y)
            if trial_f < best_fi:
                best_xi, best_yi, best_fi = trial_x, trial_y,
trial_f
            jumps_left = max_jump
        else:
            jumps_left -= 1

        self.xi, self.yi, self.fi = best_xi, best_yi, best_fi
        return self.fi
```

*Листинг E.2 — Продолжение листинга E.1*

```
def make_global_jump(self, c_x, c_y):
    self.x = self.x + np.random.uniform(1, 2) * (c_x - self.x)
    self.y = self.y + np.random.uniform(1, 2) * (c_y - self.y)

    self.x = np.clip(self.x, -4.5, 4.5)
    self.y = np.clip(self.y, -4.5, 4.5)

    self.calc()

def update(self):
    self.x, self.y, self.f = self.xi, self.yi, self.fi

class Troop:
    def __init__(self):
        self.n_monkeys = 15
        self.max_local_jumps = 20

    def monkey_init(self, minZ, maxZ):
        monkeyz = []
        for _ in range(self.n_monkeys):
            m = Monkey()
            m.x = np.random.uniform(minZ, maxZ)
            m.y = np.random.uniform(minZ, maxZ)
            m.calc()

            monkeyz.append(m)
        return monkeyz

    def find_best(self, iter):
        monkeyz = self.monkey_init(-4.5, 4.5)

        for i in range(iter):
            c_x = np.mean([m.x for m in monkeyz])
            c_y = np.mean([m.y for m in monkeyz])

            for monk in monkeyz:
                if monk.make_local_jump(self.max_local_jumps) <
monk.f:
                    monk.update()
                else:
                    monk.make_global_jump(c_x, c_y)
                    c_x = np.mean([m.x for m in monkeyz])
                    c_y = np.mean([m.y for m in monkeyz])

        monkeyz.sort(key=lambda b: b.f)
        return monkeyz[0]
```

*Листинг E.3 — Продолжение листинга E.2*

```
if __name__ == '__main__':  
    print("Эталон: f(3, 0.5) = 0")  
  
    troop = Troop()  
    iter = 100  
  
    best_monkey = troop.find_best(iter)  
    print(f"Результат x = {best_monkey.x:.5f} y =  
{best_monkey.y:.5f} f = {best_monkey.f:.8f}")
```